Università degli Studi di Napoli
"Parthenope"

Facoltà di Scienze e Tecnologie
Corso di Laurea in Informatica

Corso di Elaborazione delle Immagini Digitali

Prof. Alfredo Petrosino

ROBUST AND ACCURATE CHANGE DETECTION UNDER SUDDEN ILLUMINATION VARIATIONS

Relazione a cura di:
Virginia Bellino - Matr. 108/1570

INDICE

Introduzione

Il rilevamento delle modifiche, comunemente detto "**change detection**", consente di rilevare cambiamenti strutturali che si verificano in immagini della stessa scena analizzando una sequenza di fotogrammi della medesima acquisiti in istanti differenti

Ma cosa fa esattamente un generico algoritmo di change detection ?
Cerchiamo di capirlo fissando l'attenzione su alcuni punti fondamentali utili per comprendere quanto verrà illustrato in seguito. Il processo di Change Detection può essere scisso in due fasi, una di pre-elaborazione ed una di effettiva valutazione dei cambiamenti.

La fase di pre-elaborazione è necessaria per correggere le eventuali differenze radiometriche e gli eventuali disallineamenti tra le immagini ed è utile per applicare trasformazioni capaci di migliorare le prestazioni dei rivelatori e diminuire i tempi di elaborazione.

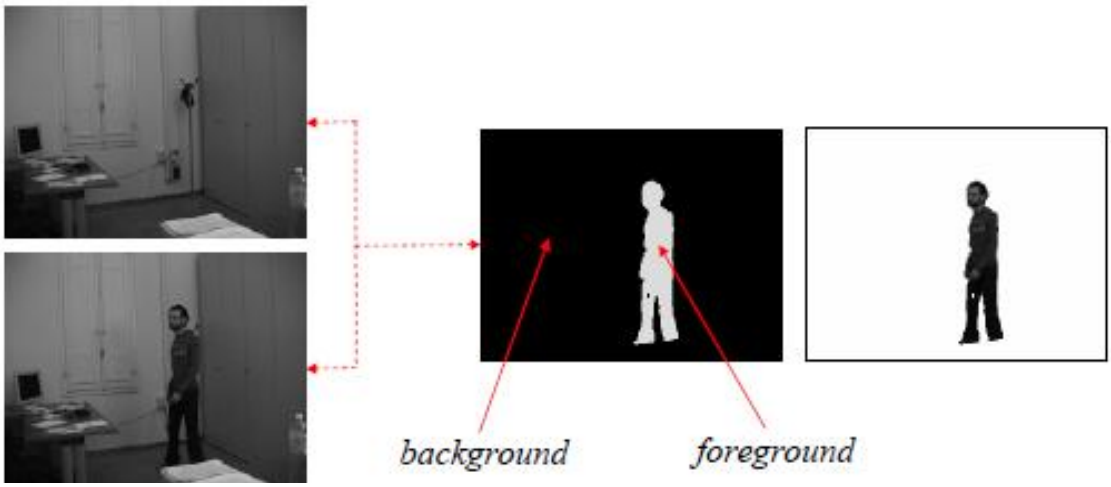
La fase successiva si basa sull'applicazione di algoritmi di rivelazione che permettono di individuare i cambiamenti occorsi nella scena.

Dunque, un algoritmo di Change Detection prevede:

- in **input** : due o piu' immagini della scena.

- in **output** : immagine binaria, detta "Change Mask", che ad ogni pixel associa uno fra due valori *c* ("changed") o *u* ("unchanged"), a seconda che il pixel presenti "cambiamenti significativi" o meno (di solito, $c = 255$, $u = 0$ → bianco/nero).

Ad esempio, considerando le immagini che seguono, applicando un algoritmo di change detection, l'obiettivo è quello di ottenere un'accurata segmentazione del foreground rispetto al background, e quindi dovremmo ottenere un risultato del tipo :



Le applicazioni degli algoritmi di change detection possono riguardare diversi ambiti, quali ad esempio la videosorveglianza ed il monitoraggio del traffico.

• Videosorveglianza.

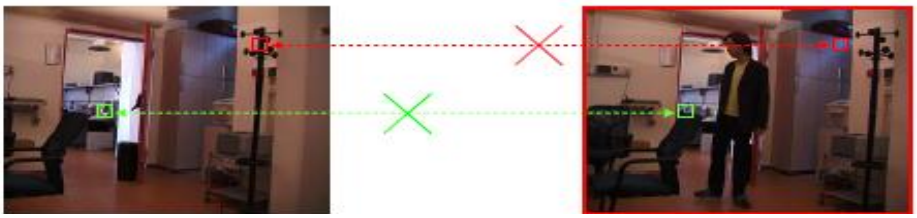
• Monitoraggio del traffico

Per quanto concerne invece le condizioni in cui opera un algoritmo di change detection, possiamo dire che i parametri considerati sono:

- **posizione della videocamera**

• **Videocamera statica:** pixel corrispondenti nelle due immagini rappresentano la stessa porzione di scena → change detection per confronto diretto.

• **Videocamera mobile:** change detection successiva alla registrazione delle due immagini.



- **frame rate (frequenza di campionamento temporale dei frame, misurata in frame/ secondo)**

• **Basso frame rate** (frequenza di campionamento temporale dei frame, frame/s): change detection per confronto fra il frame corrente ed un frame precedente.

• **Alto frame rate** (tipicamente > 1 frame/s): change detection per confronto fra il frame corrente ed un insieme di frame precedenti (analizzabile la dinamica dei cambiamenti).

Descrizione del contenuto della relazione

Uno dei compiti più impegnativi per la maggior parte delle applicazioni di “change detection “ è quello di segmentare con precisione il foreground dallo sfondo in presenza di pesanti modifiche dell’illuminazione. Infatti, le modifiche fotometriche possono essere facilmente interpretate in modo erroneo come cambiamenti strutturali, portando così ad avere falsi positivi nella maschera di modifica (“ change mask”) ottenuta in output.

Nella presente relazione viene descritto e testato un algoritmo, sviluppato presso il COMPUTER VISION LABORATORY dell’ Università di Bologna, volto ad ottenere una robusta e precisa segmentazione del foreground in presenza di improvvise variazioni di illuminazione.

Dopo aver collocato il problema appena descritto nella letteratura corrente, nelle sezioni successive viene descritta nel dettaglio la strategia proposta dagli autori, strategia che poi è stata implementata con una Dynamic Link Library (estensione . dll) denominata “ **MF2lib.dll** “.

Nella sezione “ consigli d’uso “ vengono forniti alcuni consigli per poter utilizzare al meglio il software, per evitare di incorrere nelle medesime difficoltà da me riscontrate in fase di primo utilizzo del software.

La sezione denominata TESTING descrive invece una serie di test effettuati sull’algoritmo.
La relazione si conclude con l’elenco degli opportuni riferimenti bibliografici.

Inquadramento del problema nella letteratura

L’approccio più comune di rilevamento delle modifiche fa riferimento alla sottrazione dello sfondo: dato un frame corrente F, ed un modello di sfondo della scena B, la maschera di modifica è ottenuta come differenza tra F e B. Questo approccio assume che il modello di sfondo sia disponibile o possa essere ottenuto elaborando una breve sequenza di frame al tempo di inizializzazione.

Un’altra metodologia utilizzata per il change detection è anche la differenza tra frame successivi, che calcola la differenza tra due o più frame presi ad istanti successivi per determinare se c’è stato, appunto, un cambiamento.

Nella letteratura del settore, sono stati proposti, nel corso del tempo, molti algoritmi di rilevamento delle modifiche, allo scopo di poter gestire nel miglior modo possibile problemi come i cambi di luminosità, e lo sfondo camuffato e vacillante.

Un grosso problema per molte applicazioni pratiche di rilevamento delle modifiche è rappresentato da improvvise variazioni di illuminazione che si verificano nella scena. Essere corretti su questo problema è un compito difficile per gli algoritmi di rilevamento delle modifiche, poiché le variazioni fotometriche risultanti possono essere facilmente male interpretate come cambiamenti strutturali, che porta a molti falsi positivi nella maschera di modifica.

Alcuni algoritmi che hanno il preciso scopo di rilevare le modifiche in modo robusto rispetto a improvvise variazioni di illuminazione, tipicamente prendono la decisione di votare un pixel come modificato o invariato sulla base di un dato supporto spaziale. In genere tali algoritmi si basano su un modello parametrico (ad esempio lineare) o non parametrico (ad esempio di mantenimento dell'ordine), per rilevare le false modifiche dell’immagine dovute ad improvvise variazioni di illuminazione.

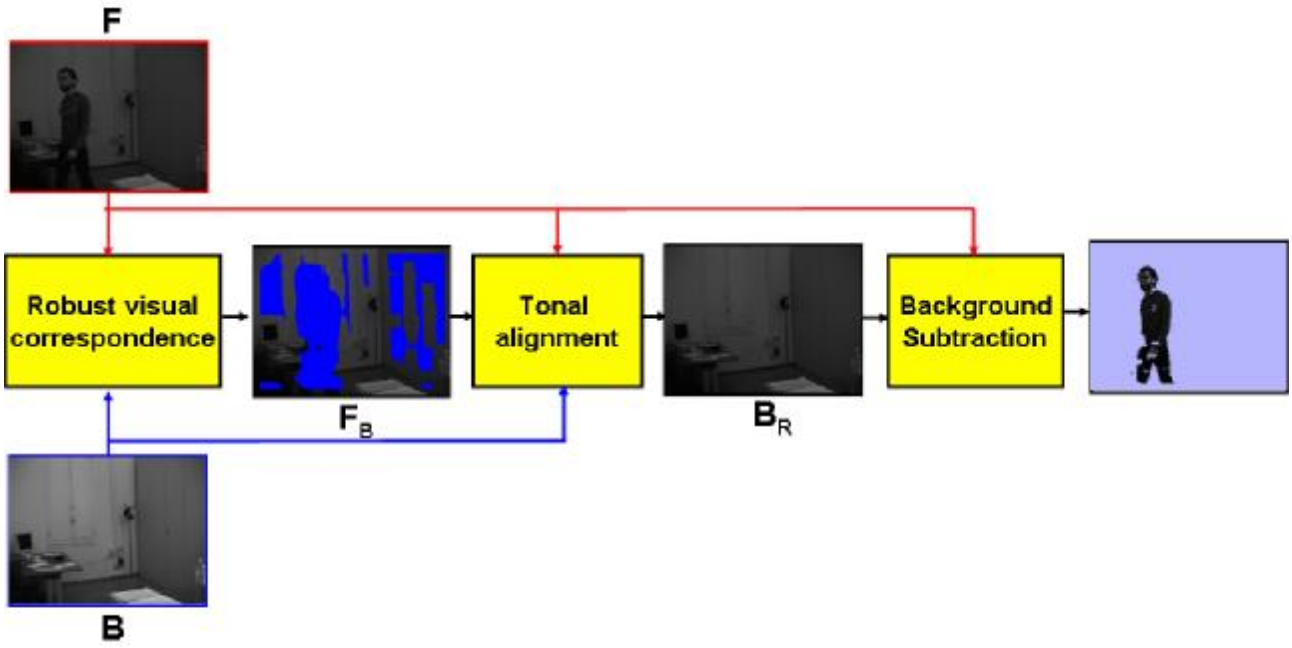
Tuttavia, è ben noto che tali algoritmi soffrono di un problema di apertura, cioè non possono rilevare come modificati i pixel appartenenti a regioni di foreground prive di texture. Come risultato, essi tipicamente consentono di rilevare i bordi degli oggetti in primo piano, ma non rilevano accuratamente le loro parti interne.

Inoltre, l'uso di un supporto spaziale al posto della sottrazione dello sfondo punto per punto implica imprecisione per quanto riguarda la localizzazione dei bordi degli oggetti in primo piano rilevati.
Naturalmente, esistono anche algoritmi che possono alleviare questi problemi

Algoritmo proposto

- Descrizione della strategia

Come rappresentato in figura ,



la strategia proposta è composta da 3 fasi di elaborazione, e cioè:

- FASE 1: CORRISPONDENZA VISUALE ROBUSTA
- FASE 2: ALLINEAMENTO TONALE
- FASE 3: SOTTRAZIONE DELLO SFONDO

Vediamo nel dettaglio ciascuna fase.

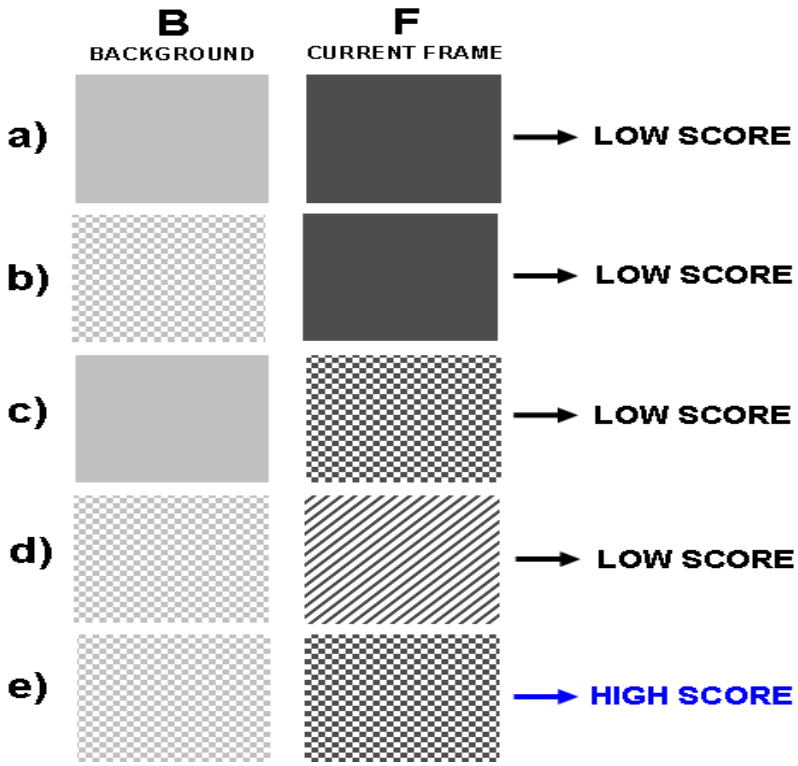
FASE 1: Corrispondenza visuale robusta

Nella prima fase, viene usata una misura della corrispondenza visuale robusta alle variazioni fotometriche per estrarre un sottoinsieme di pixel dal frame corrente. Tale sottoinsieme può essere etichettato come sfondo corrente con un alto livello di sicurezza.

Più in dettaglio, chiameremo **Fb** il sottoinsieme di pixel etichettato come sfondo corrente. Per ottenere Fb sostanzialmente abbiniamo i punti dell'immagine di background (che nel nostro caso corrisponde al primo frame della sequenza, e viene calcolato dalla prima delle due funzioni che compongono la DLL) al frame corrente, usando una strategia basata sui blocchi.

Tale strategia prevede che, per ogni coppia di punti corrispondenti nello sfondo B e nel frame corrente F , venga considerato un blocco circostante di dimensione M × M , e un punteggio di corrispondenza è calcolato tra i due blocchi. I punti che danno luogo ad un punteggio più alto di una determinata soglia sono inclusi nello sfondo corrente Fb.

Dato che il compito della prima fase è quello di trovare un insieme di punti in F che possa essere attendibilmente classificato come sfondo corrente anche in presenza di forti variazioni di illuminazione, dobbiamo determinare una misura di corrispondenza adeguata per confrontare i blocchi. A questo scopo, osserviamo la figura che segue, la quale presenta una serie di possibili ipotesi da considerare.



In tale figura, per semplicità, si considerano solo due tipi di regioni:

- uniforme
- altamente strutturata (highly textured) .

Quando si tratta di una regione uniforme sia in F che in B (***caso a*** in figura), le differenze fotometriche tra F e B possono essere dovute sia a variazioni delle condizioni di illuminazione della scena di sfondo sia alle variazioni strutturali indotte da un oggetto uniforme in primo piano. Così, in questo caso la misura di corrispondenza richiesta dovrebbe produrre un punteggio basso, e non si può dire attendibilmente se il punto appartiene allo sfondo oppure no.

Per quanto riguarda i ***casi b, c, d,*** è facile osservare che il punteggio di corrispondenza dovrebbe essere basso anche qui, poiché c'è prova della presenza di un oggetto in primo piano.

Infine, quando lo sfondo è molto strutturato e la trama del pattern ("texture") non cambia a dispetto delle possibili modifiche fotometriche (caso e), è ragionevole evidenziare il punto come sfondo con un elevato livello di sicurezza. Quindi, nel ***caso e*** dovremmo ottenere un punteggio elevato dalla misura di corrispondenza richiesta.

Sulla base delle considerazioni di cui sopra, viene adottata una misura di corrispondenza visuale recentemente proposta , chiamata qui **MF** (**matching function – funzione di corrispondenza**). Questa misura abbina i blocchi corrispondenti delle 2 immagini controllando implicitamente un vincolo d'ordine.

Poiché le variazioni fotometriche (cioè i cambi di illuminazione) tendono a non violare l'ordinamento di intensità in una zona di pixel, la MF permette di gestire improvvise e forti variazioni di illuminazione tra la scena di sfondo e il frame corrente. In particolare, siano F e B funzioni di 2 coordinate (i,j) .

Poi, una volta definito un vettore di differenze di pixel calcolato ad un punto (i, j) su F:

$$\delta_F(i,j) = F(i-1,j) - F(i+1,j) \quad (1)$$
$$F(i,j-1) - F(i,j+1)$$

e similmente , al punto (i , j) su B :

$$\delta_B(i,j) = B(i-1,j) - B(i+1,j) \quad (2)$$
$$B(i,j-1) - B(i,j+1)$$

la funzione **MF** calcolata nel blocco centrato alle coordinate (x, y) è definita come :

$$MF(x,y) = N(x,y)/D_F(x,y) \circ D_B(x,y) \quad (3)$$

dove ° rappresenta il prodotto puntuale tra due vettori e

$$N(x,y) = \sum\sum \delta_F(x+i, y+j) \circ \delta_B(x+i, y+j) \quad (4)$$

Quindi, questa funzione calcola la correlazione normalizzata tra le differenze di intensità dei pixel adiacenti, la normalizzazione al denominatore di (3) considerando la misura nell'intervallo [-1,1].

Come precedentemente introdotto, una volta che MF (x, y) viene calcolata per tutte le coordinate dei punti (x, y) di F e B, si adotta una soglia in modo che solo i punti con un punteggio superiore ad un certo valore sono inclusi in Fb.

Si può affermare facilmente che la MF cerca di far corrispondere le regioni ad alto contrasto (cioè i bordi intensità) dei due blocchi a confronto, dato che solo le differenze ad alta intensità possono fornire elevati contributi al punteggio di correlazione.
Quindi la MF si comporta esattamente come indicato nella figura vista ad inizio sezione.

Infatti, solo due modelli altamente strutturati e altamente correlati possono fornire un alto punteggio di corrispondenza (caso e), mentre la presenza di almeno una regione non strutturata (untextured) (casi a, b, c) o di due modelli strutturati ma non correlati (caso d) produce un punteggio basso.

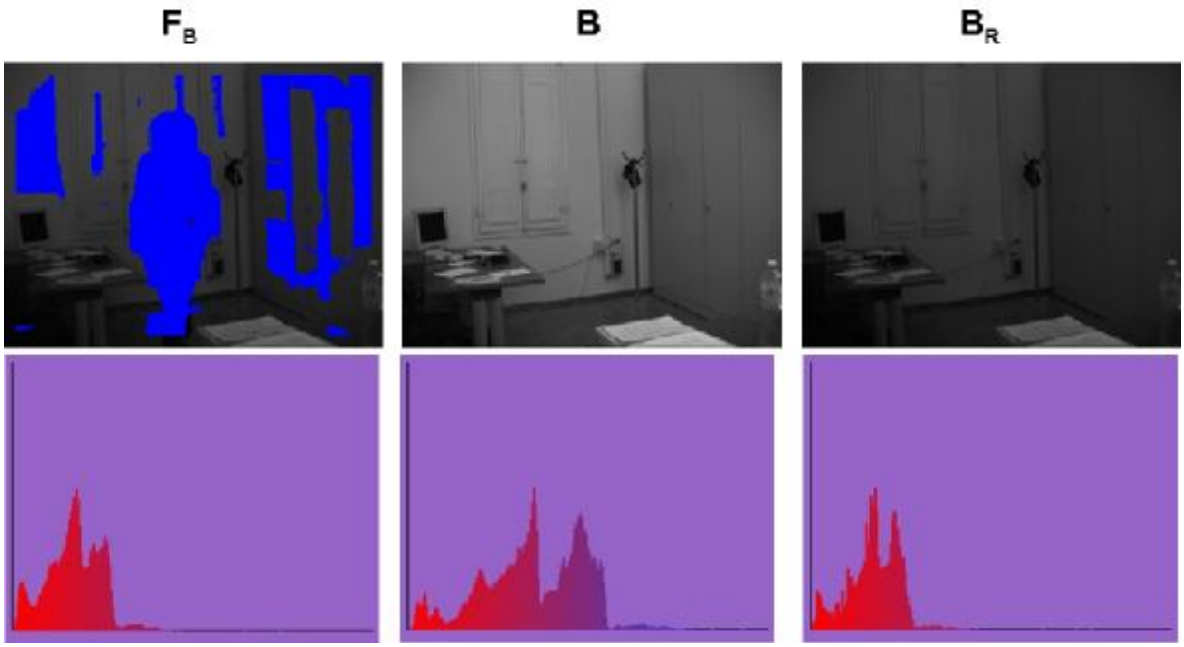
FASE 2: Allineamento tonale

Una volta che il sottoinsieme **Fb**, è stato ottenuto, può essere utilmente impiegato per rimuovere le alterazioni fotometriche tra F e B. A questo scopo, nella seconda fase l'algoritmo calcola la trasformazione che allinea tonalmente il frame corrente F alla immagine di sfondo B, usando come supporto il sottoinsieme Fb.

L'allineamento tonale di B ad F viene effettuato applicando il **metodo di specifica dell'istogramma** (conosciuto anche come " **matching tra istogrammi** "). Questo metodo

consente di ottenere una funzione di trasformazione che useremo per effettuare l'allineamento tonale, e che chiamiamo IMF (Intensity Mapping Function).

Nella valutazione della IMF che allinea B ad F, solo l'insieme dei punti corrispondenti che appartengono alla maschera Fb viene preso in considerazione. . Dunque, nella seconda fase, lo sfondo corrente Fb viene allineato tonalmente al modello di sfondo B, azione che poi consentirà di effettuare la sottrazione dello sfondo e l'accurata segmentazione del foreground nella fase successiva. Applicando infatti la IMF ottenuta dal metodo di specifica dell'istogramma a B otteniamo un nuovo sfondo Br , in cui le distorsioni fotometriche sono state rimosse.
Tale sfondo sarà poi usato nella terza fase per ottenere la segmentazione del foreground ed effettuare cosi il change detection.



[La fase di allineamento tonale registra, sulla base dell'istogramma specificato da Fb (a sinistra), l'istogramma dello sfondo B (centro), ottenendo lo sfondo tonalmente registrato BR (destra)]

FASE 3: Sottrazione dello sfondo

Nella terza fase, una semplice differenza pixel per pixel tra Br ed F evidenzia le modifiche strutturali estraendo correttamente le regioni in primo piano

È importante sottolineare che, poiché la sottrazione dello sfondo avviene pixel per pixel, non è influenzata dal problema dell'apertura e consente di rilevare con precisione i confini nonché parti interne di oggetti in primo piano.
Ovviamente, i falsi negativi possono ancora essere trovati a causa della possibile mimetizzazione tra lo sfondo tonalmente registrato e gli oggetti in primo piano.

- Requisiti computazionali

Per quanto riguarda invece i **requisiti computazionali**, si può osservare che il collo di bottiglia dell'algoritmo proposto può essere identificato nella prima fase.

Infatti, denotando come W ed H rispettivamente la larghezza e l'altezza delle immagini, il calcolo di N (4) richiede teoricamente 2*(M^2)*W*H differenze, moltiplicazioni e sommatorie, e similmente Df (5) e Db (6) richiedono ciascuno 2*(M^2)*W*H differenze, moltiplicazioni e sommatorie più W*H radici quadrate.

Tuttavia, tutte le differenze riguardanti il modello di sfondo B possono essere calcolate, una volta per tutte, in fase di inizializzazione, mentre tutte le differenze riguardanti il frame corrente possono essere calcolata, una volta per tutte, ogni nuovo fotogramma, tenendo conto di un totale di 2*W*H differenze. Inoltre, poiché la metrica corrispondente deve essere applicato su blocchi vicini, approcci di tipo incrementale consentono un'ulteriore contrazione del numero totale di operazioni necessarie per il calcolo di N.

In particolare, questo può essere fatto calcolando il prodotto delle differenze di pixel corrispondenti, una volta per tutte ad ogni nuovo fotogramma (tenendo conto di 2*W*H moltiplicazioni) , poi eseguendo due istanze di Filtro Box per calcolare il termine finale di raccolta, che rappresenta 8*W*H sommatorie complessive.
Deduzioni analoghe valgono per il calcolo dei due termini denominatore, Df e Db. Quest'ultimo può essere calcolato una volta per tutte in fase di inizializzazione, mentre, per mezzo di una strategia analoga a quella utilizzata per calcolare N, il primo richiede solo 2*W*H moltiplicazioni aggiuntive, 8*W*H sommatorie e W*H radici quadrate complessivamente ad ogni nuovo fotogramma.

Grazie a queste ottimizzazioni, l'implementazione dell'algoritmo proposto ha facilmente attinenza con i requisiti in tempo reale di molte applicazioni di rilevamento delle modifiche (ad es videosorveglianza) , con una frequenza di fotogrammi pari a 15 fps (fotogrammi al secondo) su una dimensione del frame di 320X240

- Implementazione tramite DLL

La strategia appena descritta viene implementata con una una libreria a collegamento dinamico (DLL) denominata **MF2lib.dll.** Tale libreria utilizza OpenCV.

La suddetta DLL comprende le seguenti funzioni di "change detection", che elaborano i frame catturati da uno stream video :

```
//la "preMF2Function" è una function che viene chiamata una volta all'inizializzazione del sistema
//ed ogni volta che cambia lo sfondo.

void preMF2Function
(
        IplImage* background, // the first frame (captured image)
        int SBL // patch radius (parameter of the algorithm)
);

//The MF2Function has to be called at every new frame of the video sequence and outputs a change
//mask ("dest"). It also needs three parameters (the parameters of the algorithm):
```

SBL, the patch radius (the same value as in the preMF2Function)
MF2Th, the threshold for the MF matching step

diffTh, the pixelwise background subtraction threshold

```
void MF2Function
(
        IplImage* currentFrame, // the current frame (captured image)
        IplImage* background, // the first frame stored in preMF2Function
        IplImage* dest, // the result of algorithm stored in a frame
        int SBL, // patch radius (parameter of the algorithm)
        float MF2Th, // MF threshold (parameter of the algorithm)
        int diffTh // Pixelwise background subtraction threshold (parameter of the algorithm)
);
```

Come si può notare osservando il codice delle 2 funzioni sopraelencate, viene fatto largo uso della struttura **IplImage**

Questa struttura dati è una struttura di base di OpenCV presente però nelle versioni molto vecchie, per la precisione nelle versioni precedenti ad OpenCV 2.2.

IplImage deriva dalle **ipl** (Intel Image Processing Library – ora non più utilizzate) ed è contenuta in **CxCore** , libreria che ha poi assunto altra veste nelle versioni di OpenCV successive alla 2.2

Altra libreria utilizzata nel programma di esempio che carica dinamicamente la dll è la libreria **CvCam**, che è stata poi deprecata nelle versioni di OpenCV successive alla 2.2

- **Parametri di input**

int SBL: dimensione in pixel della finestra utilizzata per il calcolo della misura di matching per determinare il background

float MF2Th: soglia intermedia usata per classificare I pixel tra background e foreground

int diffTh: soglia finale dell'algoritmo, usata per fare la sottrazione dello sfondo pixel per pixel. Essa definisce un punto di lavoro preciso, in termini di falsi positivi e veri positivi.

Tipicamente, per valutare gli algoritmi di change detection si fa scorrere questa soglia da un valore minimo ad un valore massimo, calcolando per ogni valore del range il punto della curva ROC (True Positive Rate vs. False Positive Rate) - o equivalentemente della curva Precision-Recall - e plottando successivamente i risultati su un grafico. Infatti, un valore conservativo della soglia significa generalmente pochi falsi positivi ma pochi veri positivi (alta precision e bassa recall), mentre un valore più ampio della soglia significa il contrario (molti falsi positivi e molta recall, dunque più alta recall ma a costo di una piu bassa precision).

- **Cos'è una dll**

A beneficio degli utenti meno esperti, illustriamo brevemente il concetto di **Dynamic Link Library**, fornendo una serie di nozioni che potranno risultare utili per comprendere alcune delle istruzioni

In informatica, una *dynamic-link library* (termine inglese, tradotto in italiano con libreria a collegamento dinamico) è una libreria software che viene caricata dinamicamente in fase di esecuzione, invece di essere collegata staticamente a un eseguibile in fase di compilazione. Queste librerie sono note con l'acronimo DLL, che è l'estensione del file che hanno nel sistema operativo Microsoft Windows, o anche con il termine librerie condivise (da *shared library*, usato nella letteratura dei sistemi Unix).

La separazione del codice in librerie a collegamento dinamico permette di suddividere il codice eseguibile in parti concettualmente separate, che verranno caricate solo se effettivamente necessarie. Inoltre, una singola libreria, caricata in memoria, può essere utilizzata da più programmi, senza la necessità di essere nuovamente caricata, il che permette di risparmiare le risorse del sistema. Questo metodo di *loading on demand* consente, inoltre, installazioni parziali di un sistema software, in cui sono effettivamente presenti sulla memoria di massa solo le librerie associate alle funzioni che l'utente desidera utilizzare, come selezionate in fase di installazione. Un altro vantaggio è la possibilità di aggiornare un programma modificando solo le DLL.

Il principale svantaggio delle DLL è legato al fatto che una nuova versione di una DLL potrebbe effettuare dei cosiddetti *breaking changes*, in modo volontario o, inconsapevolmente, a causa di bug nella nuova versione.

Un breaking change è un cambiamento critico nel comportamento del codice della funzione che la rende non più compatibile con le convenzioni in uso (ad esempio, una funzione che prima restituiva NULL in caso di errore nei parametri e che ora setta *errno* e restituisce un valore non nullo).

> *Nota:*
> *Nel caso esaminato, la DLL MF2lib.dll implementa funzioni che utilizzano tipi di dati presenti nelle vecchie versioni di OpenCV (cersioni precedenti alla 2.2), come ad esempio la struttura* ***IplImage*** *, e questo porta ad avere errori in fase di compilazione se si usa una versione di OpenCV superiore alla 2.2.*

Una libreria a collegamento dinamico è a tutti gli effetti un codice eseguibile. Ogni file eseguibile (EXE o DLL) dispone di un **punto d'ingresso** (***entry point***) invocato dal sistema operativo subito dopo il caricamento. Per una DLL il punto d'ingresso è mappato per convenzione sulla funzione DllMain (a discrezione, comunque, del compilatore).

La funzione DllMain, oltre che al caricamento della DLL, viene invocata anche allo scaricamento o quando un thread viene creato o distrutto nel processo in cui la DLL risiede.

A differenza di un file EXE, la DLL deve uscire dall'entry point non appena ha terminato le inizializzazioni necessarie.

Per semplificare, da un punto di vista strutturale possiamo affermare che una libreria può essere pensata come una raccolta di funzioni.

Ognuna di queste funzioni avrà il proprio indirizzo di base, calcolato come offset rispetto all'indirizzo di base assegnato dal sistema operativo durante il caricamento della libreria. Ciò che distingue una libreria dinamica è che queste funzioni possono essere *esportate*, ovvero i loro nomi vengono posti in una lista in una sezione dell'eseguibile. Perciò è possibile determinare il punto di ingresso di una funzione con una ricerca testuale basata sul nome della funzione. Questa operazione è svolta dall'API **GetProcAddress** che restituisce l'indirizzo della funzione il cui nome è passato come parametro.

Le librerie dinamiche vengono caricate dal sistema operativo all'interno dello spazio di memoria del processo che le ha richieste. In questo modo l'accesso al codice della DLL avrà prestazioni quasi equivalenti a quelle del codice dell'applicazione stessa o del codice delle librerie statiche.

Per evitare che il codice dell'applicazione e quello della DLL occupino la stessa posizione in memoria, il linker dovrà predisporre la DLL per la rilocazione. In pratica, il sistema operativo determina un'area di memoria disponibile e rimappa ogni riferimento alla memoria contenuto nel codice della DLL. Siccome quest'operazione richiede tempo, ogni DLL dispone di un proprio *indirizzo di base* ideale: la rilocazione sarà necessaria solo se a questo indirizzo predeterminato è già stata mappata una precedente DLL.

Il collegamento di un eseguibile a una libreria dinamica avviene durante l'esecuzione (a *run time*) e avviene tramite l'API LoadLibrary, che accetta in input il nome della libreria. Ad esempio, LoadLibrary(_T("MyLib.dll")) caricherà all'interno dello spazio di memoria dell'applicazione la DLL MyLib.dll.

Il collegamento può essere di due tipi: esplicito o implicito

- il collegamento esplicito viene gestito direttamente dal codice del programma con l'utilizzo delle due API `LoadLibrary` e `GetProcAddress` .

  Questa tecnica permette di gestire in modo appropriato la condizione nella quale una DLL richiesta non è presente nel sistema, ma in generale è più macchinosa perché richiede l'utilizzo *esplicito* delle due API

- Il collegamento implicito è gestito direttamente dal linker in fase di compilazione, ed è usato quando si assume che una DLL sia sempre presente nel sistema.

Consigli d'Uso

La presente sezione illustra alcune problematiche da me riscontrate in fase di primo utilizzo del software, e mostra le relative soluzioni adottate, spiegando, ove possibile, il motivo per cui si ottengono determinati risultati.

Per poter effettuare i test ho creato due ambienti di lavoro.

Nel 1° caso, l'ambiente è costituito da:

- visual studio 2012 professional, su sistema operativo WINDOWS 7 starter

- openCV 1.0 (il motivo di questa scelta risiede nel fatto che, studiando il programma di esempio fornito sul sito web del software, ovvero

  http://labvisione.deis.unibo.it/software/94-mf-cd

  e guardando la data di ultima modifica della DLL (10 febbraio 2009), poi ricompilata dall'autore in un momento successivo (26 maggio 2014) , ho ipotizzato che fosse questa la versione usata per creare la medesima. Per fugare eventuali dubbi ho anche provato ad includere nei test da me effettuati versioni di opencv post 2.0, ricevendo però errori in compilazione

Nel 2° caso, l'ambiente è costituito da:

- visual studio 2010 professional, su sistema operativo WINDOWS XP

- openCV 1.0

Si consuglia di utilizzare Visual Studio 2010 come ambiente di sviluppo perché con il 2012 si possono avere dei problemi in fase di compilazione dovuti probabilmente ad incompatibilità con qualche libreria specifica.

La DLL è stata infatti compilata con Visual Studio 2010.

Testing

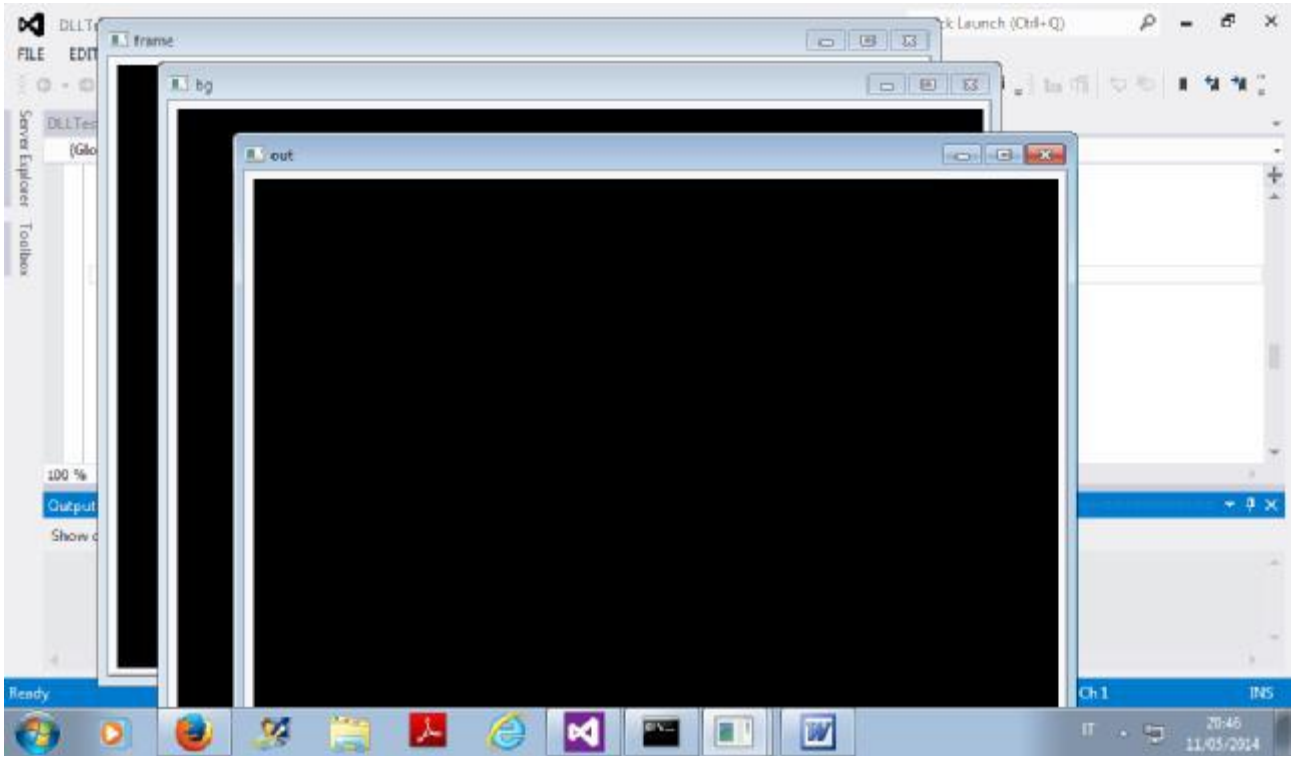
- Esempio 1: input da webcam

```
// DLLTest.cpp : Defines the entry point for the console application.
// A sample test to use MF2lib.dll by Edoardo Sabatini for CVLab.
//
// Note: dynamic loading the MF2lib.dll is the harder way.
// This application needs to include Windows.h for HINSTANCE
// and loading MF2lib.dll and releasing functions.
//

#include "stdafx.h"
#include <iostream>
#include <windows.h>

// This application needs to include cv.h for OpenCV functions

#include "cv.h"
#include "cvaux.h"
#include "cxcore.h"
#include "highgui.h"
#include "cvcam.h"

/* Function pointers that will be used for the MF2lib.dll functions. */

typedef void (*preMF2Function)(IplImage*, int);
typedef void (*MF2Function)(IplImage*, IplImage*, IplImage*,
                                                int, float, int);

// Must have a main function to try the MF2lib.dll

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout << "DLLTest v. 1.00 - @2009 CVLab" << std::endl;

       // Typedef functions to hold what is in the MF2lib.dll
       // servono a contenere quanto è nella MF2lib.dll

       preMF2Function _preMF2Function;
       MF2Function _MF2Function;

       // The Instance of the MF2lib.dll
       // LoadLibrary used to load the MF2lib.dll

       // LoadLibrary è l'API che carica materialmente la DLL
       // HiNSTANCE è un tipo di dati in Windows. Contiene l'istanza della DLL caricata
       dinamicamente
       HINSTANCE hinstanceMF2lib = LoadLibrary(L"MF2lib.dll");

       if(hinstanceMF2lib)
       {
              std::cout << "Mf2lib.dll loaded." << std::endl;

              // Our MF2lib.dll is loaded and ready to go.
              // Set up our function pointers.

              // trova l'esatta posizione in memoria delle funzioni della DLL
              _preMF2Function = (preMF2Function)
                     GetProcAddress(hinstanceMF2lib, "preMF2Function");
              _MF2Function = (MF2Function)
                     GetProcAddress(hinstanceMF2lib, "MF2Function");
```

```
// Check if _preMF2Function is currently holding
// a function, if not don't run it.

//verifica se le istanze delle 2 funzioni della DLL sono pronte all'uso
if(_preMF2Function) std::cout << "preMF2Function held!" << std::endl;

// Check if _MF2Function is currently holding
// a function, if not don't run it.

if(_MF2Function) std::cout << "MF2Function held!" << std::endl;

// Wait for the user to press enter to exit
// si attende che l'utente prema INVIO per iniziare il test
std::cout << "Press Enter to start test." << std::endl;
std::cin.get();

// parameters of algorithm

int SBL = 5;
float MF2Th = 0.6;
int diffTh = 20;
std::cout << "Parameters of algorithm:" << std::endl;
std::cout << "\tSBL = " << SBL << std::endl;//patch radius
//soglia per la matching function. Serve per applicare il 1°step della
strategia
std::cout << "\tMF2Th = " << MF2Th << std::endl;
//soglia che consente di effettuare la sottrazione dello sfondo pixel per
pixel
std::cout << "\tdiffTh = " << diffTh << std::endl;

// it captures images from WebCam
// puntatore che fa riferimento allo stream video catturato
CvCapture *cap;
cap = cvCaptureFromCAM(0); //apre lo stream video e lo associa a cap
//IplImage: struttura atta a contenere tutte le informazioni dell'immagine
come dimensioni, canali, profondità pixel etc.
IplImage* img;

//comanda il device a scattare un'istantanea di quanto ripreso in un dato
momento.
//L'immagine è però nascosta.
//cvRetrieveFrame consente di ottenere l'immagine catturata al frame
precedente
cvGrabFrame(cap);
img = cvRetrieveFrame(cap);

//crea un output compatibile con il frame catturato
IplImage *background = cvCreateImage(cvGetSize(img), 8, 1);
IplImage *imgC = cvCreateImage(cvGetSize(img), 8, 3);
IplImage *frame = cvCreateImage(cvGetSize(img), 8, 1);
IplImage *cm = cvCreateImage(cvGetSize(img), 8, 1);

cvFlip(img, imgC);//gira l'immagine
//converte l'immagine in scala di grigio
cvCvtColor(imgC, background, CV_BGR2GRAY);
//carica la prima delle 2 funzioni previste dalla dll. Tale funzione viene
chiamata una volta
//all'inizializzazione del sistema e ogni volta che muta lo sfondo.
//salva il primo frame
_preMF2Function(background, SBL);

cvNamedWindow("out", 1);
cvNamedWindow("frame", 1);
```

```
            cvNamedWindow("bg", 1);
            cvShowImage("bg", background);

            int key = 'a';

            while(key != 'q'){

                  cvGrabFrame(cap);
                  img = cvRetrieveFrame(cap);

                  cvFlip(img, imgC);
                  cvCvtColor(imgC, frame, CV_BGR2GRAY);
                  // applica la seconda delle 2 funzioni previste nella DLL
                  // tale funzione viene chiamata ad ogni nuovo frame della video
                  sequenza e restituisce in output
                  // una maschera di modifica
                  _MF2Function(frame, background, cm, SBL, MF2Th, diffTh);
                  // restituisce in output il frame corrente e la maschera di modifica
                  cvShowImage("frame", frame);
                  cvShowImage("out", cm);
                  key = cvWaitKey(10);

            }// while

      } // hinstanceMF2lib
      else
      {
       // Our MF2lib.dll failed to load!
       std::cout << "MF2lib.dll Failed To Load!" << std::endl;
      }
      return 0;
} // end main
```

Output:



Come si può notare, nel test da me effettuato, l'esecuzione dell'algoritmo con input da webcam genera in output 3 schermate completamente nere.

Ciò è dovuto al fatto che la versione di OpenCV 1.0 è incompatibile con i driver di molte webcam recenti (ed infatti il PC usato per il test è un netbook del 2012), e questa peculiarità non causa messaggi di errore ma semplicemente una visualizzazione delle immagini in input completamente nera.

Finestre di output:

- **BG**: mostra il background, corrispondente al 1° frame dello stream
- **FRAME**: visualizza in frame corrente dello stream video
- **OUT**: risultato dell'algoritmo.
  Si tratta di una immagine binaria, detta "Change Mask", che ad ogni pixel associa uno fra due valori *c* ("changed") o *u* ("unchanged"), a seconda che il pixel presenti "cambiamenti significativi" o meno (di solito, *c = 255*, *u = 0* → bianco/nero).

- Esempio 2: input da file video AVI

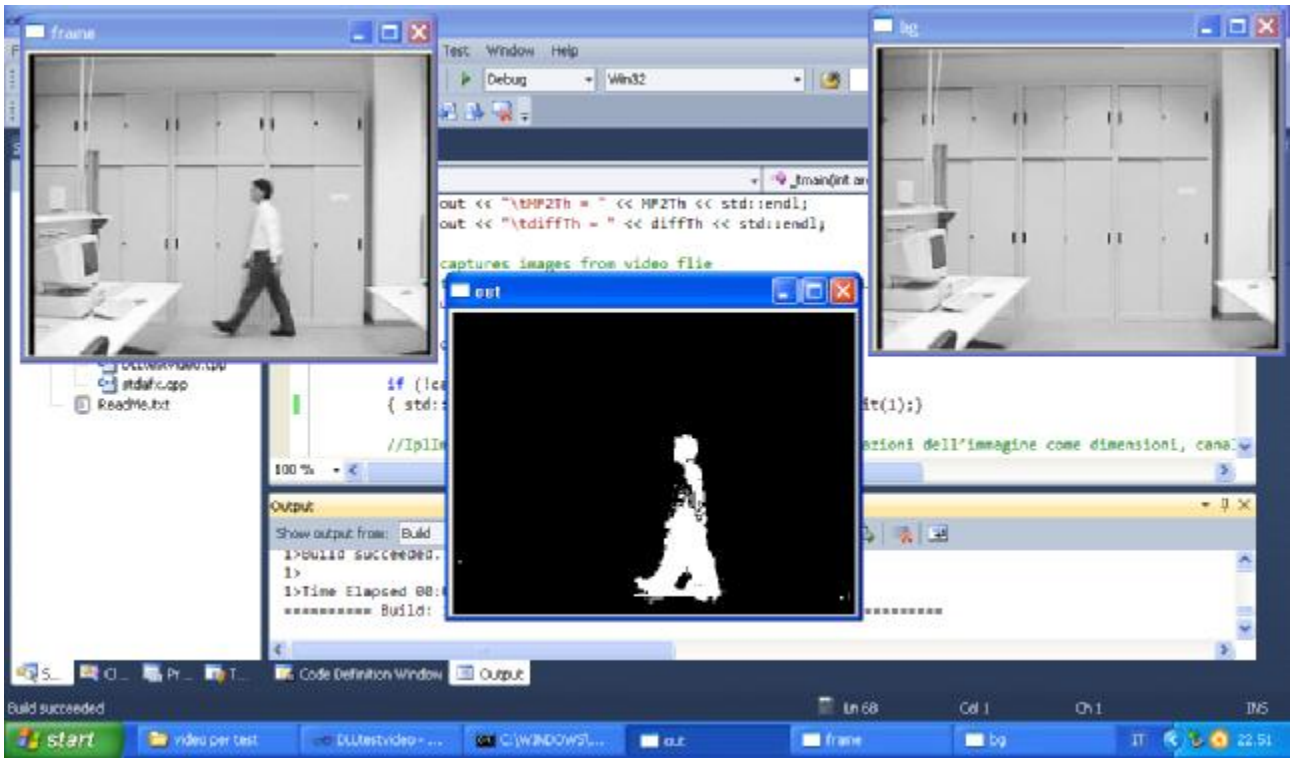
```
// DLLTest.cpp : Defines the entry point for the console application.
// A sample test to use MF2lib.dll by Edoardo Sabatini for CVLab.
//
// Note: dynamic loading the MF2lib.dll is the harder way.
// This application needs to include Windows.h for HINSTANCE
// and loading MF2lib.dll and releasing functions.
//

#include "stdafx.h"
#include <iostream>
#include <windows.h>

// This application needs to include cv.h for OpenCV functions

#include "cv.h"
#include "cvaux.h"
#include "cxcore.h"
#include "highgui.h"
//#include "cvcam.h"

/* Function pointers that will be used for the MF2lib.dll functions. */

typedef void (*preMF2Function)(IplImage*, int);
typedef void (*MF2Function)(IplImage*, IplImage*, IplImage*,
                                                int, float, int);

// Must have a main function to try the MF2lib.dll

int _tmain(int argc, _TCHAR* argv[])
{
    std::cout << "DLLTest v. 1.00 - @2009 CVLab" << std::endl;

       // Typedef functions to hold what is in the MF2lib.dll
```

```
preMF2Function _preMF2Function;
MF2Function _MF2Function;

// The Instance of the MF2lib.dll
// LoadLibrary used to load the MF2lib.dll

HINSTANCE hinstanceMF2lib = LoadLibrary(L"MF2lib.dll");

if(hinstanceMF2lib)
{
	std::cout << "Mf2lib.dll loaded." << std::endl;

	// Our MF2lib.dll is loaded and ready to go.
	// Set up our function pointers.

	_preMF2Function = (preMF2Function)
		GetProcAddress(hinstanceMF2lib, "preMF2Function");
	_MF2Function = (MF2Function)
		GetProcAddress(hinstanceMF2lib, "MF2Function");

	// Check if _preMF2Function is currently holding
	// a function, if not don't run it.

	if(_preMF2Function) std::cout << "preMF2Function held!" << std::endl;

	// Check if _MF2Function is currently holding
	// a function, if not don't run it.

	if(_MF2Function) std::cout << "MF2Function held!" << std::endl;

	// Wait for the user to press enter to exit

	std::cout << "Press Enter to start test." << std::endl;
	std::cin.get();

	// parameters of algorithm

	int SBL = 5;
	float MF2Th = 0.6;
	int diffTh = 20;
	std::cout << "Parameters of algorithm:" << std::endl;
	std::cout << "\tSBL = " << SBL << std::endl;
	std::cout << "\tMF2Th = " << MF2Th << std::endl;
	std::cout << "\tdiffTh = " << diffTh << std::endl;

	// it captures images from video flie
	// puntatore che fa riferimento allo stream video catturato
	CvCapture *cap;
	//
	cap = cvCreateFileCapture("nome del video");

	//if (!cap)
	//{ std::cout << "Couldnt get a video" << std::endl; exit(1);}

	//IplImage: struttura atta a contenere tutte le informazioni dell'immagine
	come dimensioni, canali, profondità pixel etc.
	IplImage *img;
	//comanda il device a scattare un'istantanea di quanto ripreso in un dato
	momento.
	//ed a restutuire il frame catturato. E' equivalente a
	//cvGrabFrame(cap);
	//img = cvRetrieveFrame(cap);
	img = cvQueryFrame(cap);
	//if (!img)
```

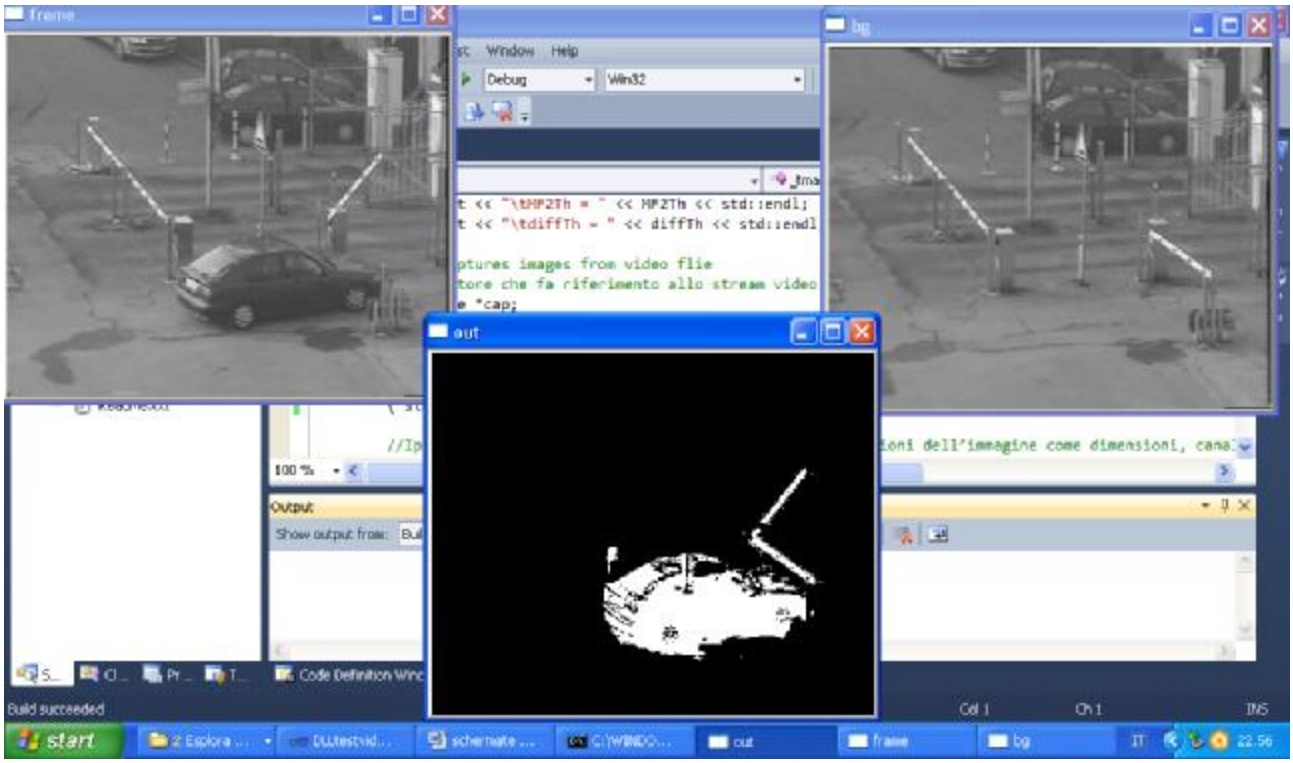
```
            //{ std::cout << "Couldnt get a camera image" << std::endl; exit(1); }

            //crea un output compatibile con il frame catturato
            CvSize img_size=cvGetSize(img);
            IplImage *background = cvCreateImage(img_size, 8, 1);
            IplImage *imgC = cvCreateImage(img_size, 8, 3);
            IplImage *frame = cvCreateImage(img_size, 8, 1);
            IplImage *cm = cvCreateImage(img_size, 8, 1);

            cvFlip(img, imgC);//gira l'immagine
            //converte l'immagine in scala di grigio
            cvCvtColor(imgC, background, CV_BGR2GRAY);
            //carica la prima delle 2 funzioni previste dalla dll. Tale funzione viene
            chiamata una volta
            //all'inizializzazione del sistema e ogni volta che muta lo sfondo.
            //salva il primo frame
            _preMF2Function(background, SBL);

            cvNamedWindow("out", CV_WINDOW_AUTOSIZE);
            cvNamedWindow("frame", CV_WINDOW_AUTOSIZE);
            cvNamedWindow("bg", CV_WINDOW_AUTOSIZE);
            cvShowImage("bg", background);

            int key = 'a';

            while(key != 'q'){

            img = cvQueryFrame(cap);

                   cvFlip(img, imgC);
                   cvCvtColor(imgC, frame, CV_BGR2GRAY);
                   // applica la seconda delle 2 funzioni previste nella DLL
                   // tale funzione viene chiamata ad ogni nuovo frame della video
                   sequenza e restituisce in output
                   // una maschera di modifica
                   _MF2Function(frame, background, cm, SBL, MF2Th, diffTh);

                   cvShowImage("frame", frame);
                   cvShowImage("out", cm);

                   key = cvWaitKey(33);

            }// while

      } // hinstanceMF2lib
      else
      {
       // Our MF2lib.dll failed to load!
       std::cout << "MF2lib.dll Failed To Load!" << std::endl;
      }
      return 0;
} // end main
```

Output:

Video n°1: Laboratory_raw.avi



Video n°2: campus_raw.avi



La funzione

```
cvCreateFileCapture("nome del video");
```

consente di acquisire lo stream video da analizzare da un file video, il cui nome viene passato come parametro alla funzione.

Bisogna però segnalare che la OpenCV 1.0 è compatibile solo con alcuni formati video, ed inoltre tale compatibilità dipende anche da quali codec risultano installati sul computer utilizzato per effettuare i test.

Negli esempi mostrati, entrambi i files video utilizzati per il test hanno estensione AVI, formato che viene gestito senza alcun problema

Finestre di output:

- **BG**: mostra il background, corrispondente al 1° frame dello stream
- **FRAME**: visualizza in frame corrente dello stream video
- **OUT**: risultato dell'algoritmo.
  Si tratta di una immagine binaria, detta "Change Mask", che ad ogni pixel associa uno fra due valori *c* ("changed") o *u* ("unchanged"), a seconda che il pixel presenti "cambiamenti significativi" o meno (di solito, $c = 255$, $u = 0 \rightarrow$ bianco/nero).

Esempio 3: test sfondo

In questo test viene mostrato che lo sfondo della scena, calcolato chiamando al tempo iniziale la prima delle 2 funzioni facenti parte della DLL, la quale salva come sfondo la prima immagine dello stream video, non subisce variazioni e viene allineato totalmente al frame corrente nella seconda fase della strategia, operazione che consentirà poi restituire la change mask nella fase finale dell'algoritmo.
Per assolvere questo compito ho modificato il main mostrato in precedenza inserendo le seguenti istruzioni:

```
//finestre che conterranno il risultato finale
cvNamedWindow("out", CV_WINDOW_AUTOSIZE); //output
cvNamedWindow("frame", CV_WINDOW_AUTOSIZE);//frame corrente
cvNamedWindow("bg", CV_WINDOW_AUTOSIZE); //sfondo iniziale
cvNamedWindow("bg2", CV_WINDOW_AUTOSIZE); //sfondo dopo la change
//detection
// differenza tra gli sfondi pre e post operazione
cvNamedWindow("differenza", CV_WINDOW_AUTOSIZE);

//mostra lo sfondo iniziale
cvShowImage("bg", background);

int key = 'a';

while(key != 'q'){
```

```
            img = cvQueryFrame(cap);

            cvFlip(img, imgC);
            cvCvtColor(imgC, frame, CV_BGR2GRAY);
            // applica la seconda delle 2 funzioni previste nella DLL
            // tale funzione viene chiamata ad ogni nuovo frame della
            video sequenza e restituisce in output
            // una maschera di modifica
            _MF2Function(frame, background, cm, SBL, MF2Th, diffTh);

            cvShowImage("bg2", background);
            cvShowImage("frame", frame);
            cvShowImage("out", cm);

            // differenza tra gli sfondi
            cvSub(background, background, diff, NULL);
            cvShowImage("differenza", diff);


            key = cvWaitKey(33);

        }// while
```

Output:

Video n°1: Laboratory_raw.avi

Video n°2: highwayI_raw.avi

Video n°3: campus_raw.avi

Esempio 4: test parametri

Nei test che seguono osserviamo come cambia la precisione dell'algoritmo al variare dei parametri di input. Nella prima serie di test proposta , lasceremo invariati i valori dei primi 2 parametri, modificando solo il valore del parametro **diffTh (soglia finale dell'algoritmo).** Questa soglia si basa essenzialmente su due misure, che forniscono il rating dei veri positivi e dei falsi positivi.
Tali misure sono:

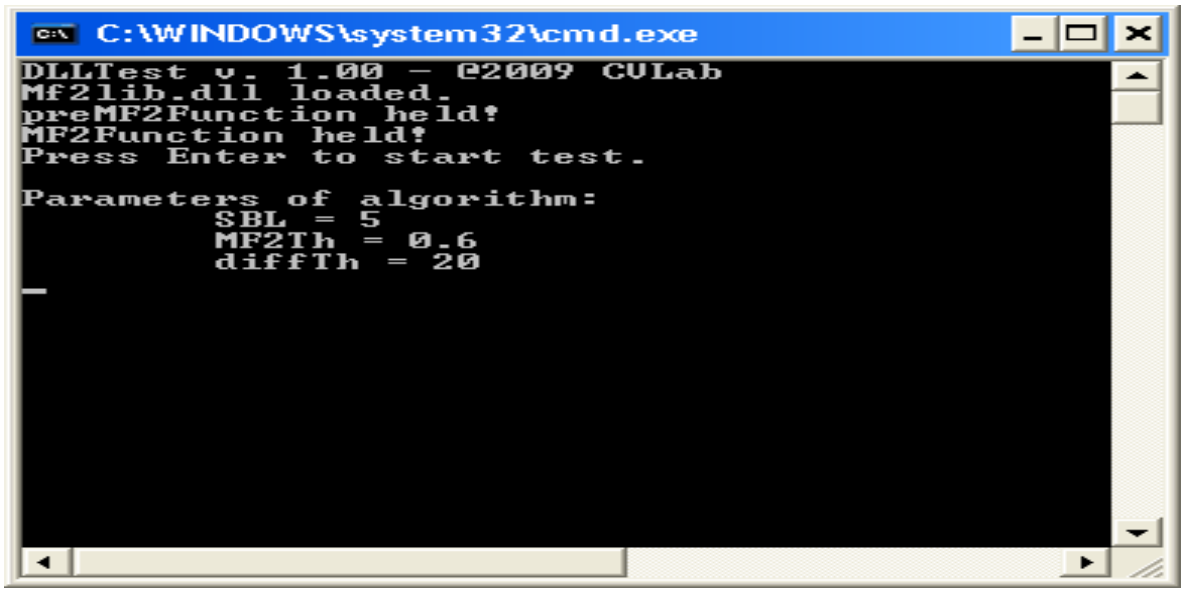
Precisione = (veri positivi) / (veri positivi + falsi positivi)

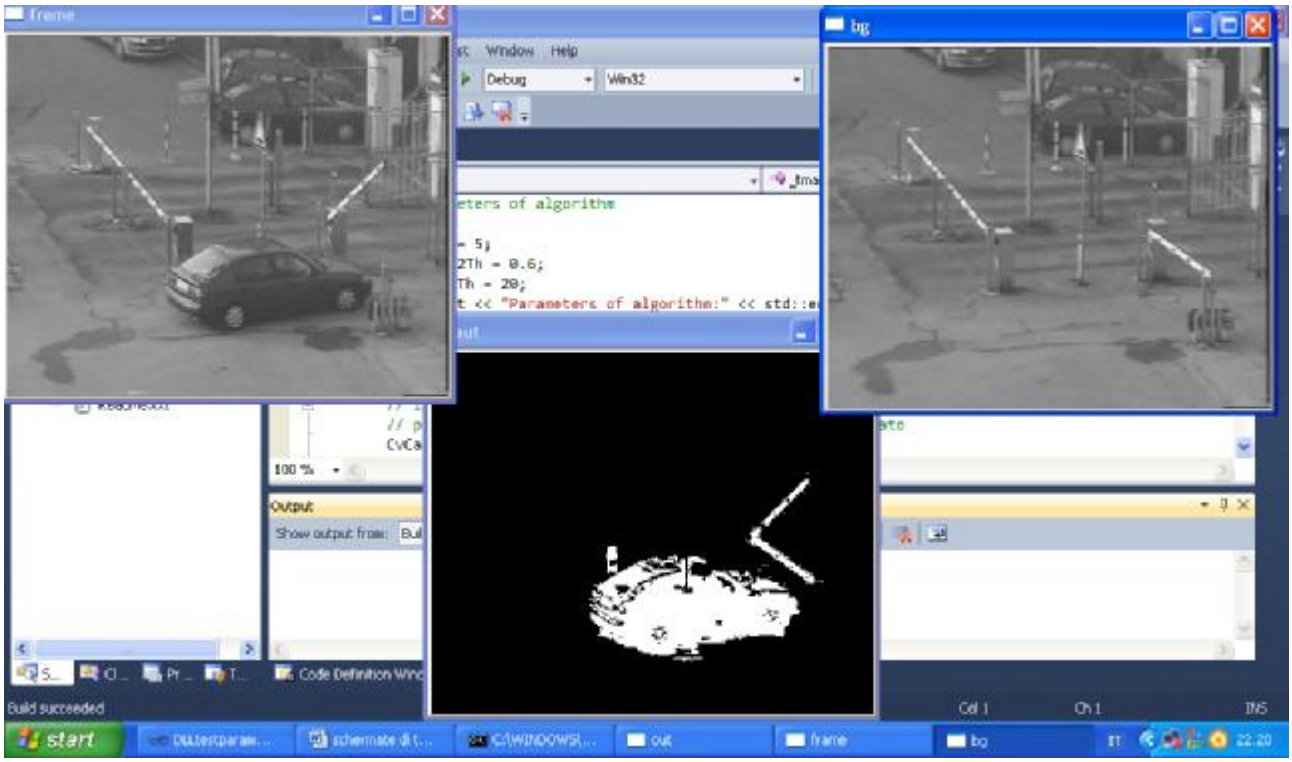
Recall = (veri positivi) / (veri positivi + falsi negativi)

Un valore basso della soglia significa significa che abbiamo pochi falsi positivi (alta precisione) ma anche pochi veri positivi e, di conseguenza, molti falsi negativi (bassa recall)
Un valore più ampio della soglia significa significa invece che i veri positivi aumentano ma aumentano anche i falsi positivi, e quindi la precisione scende.

***NOTA**: per mostrare come cambia la precisione dell'algoritmo al variare della soglia, nei test viene usato sempre il medesimo video ed in particolare verrà mostrato sempre il medesimo frame, in modo che l'osservatore possa rendersi conto anche graficamente del mutamento di precisione dell'algoritmo.*

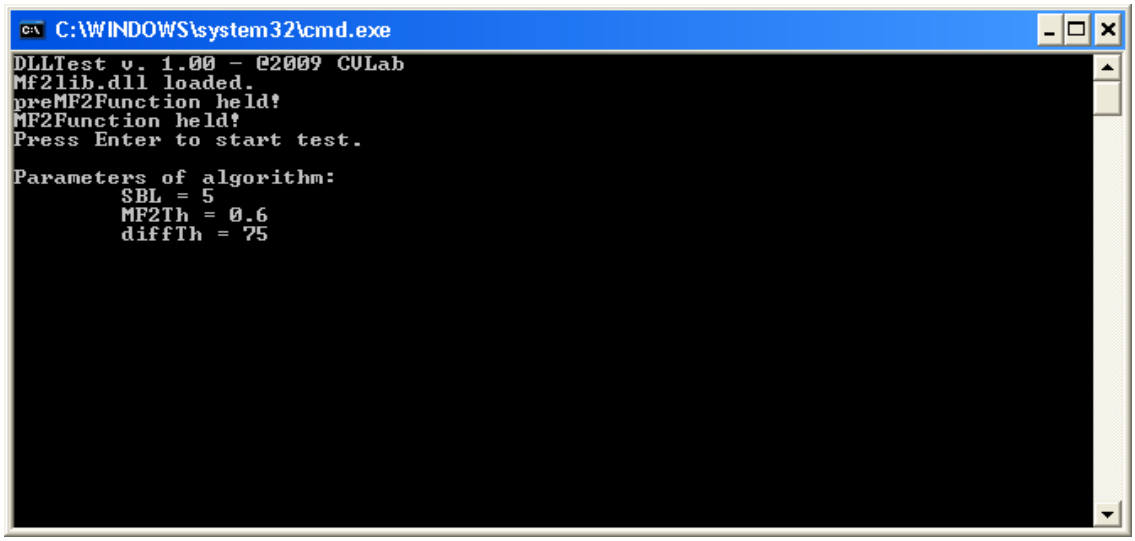
1) Valori di default forniti dall'autore

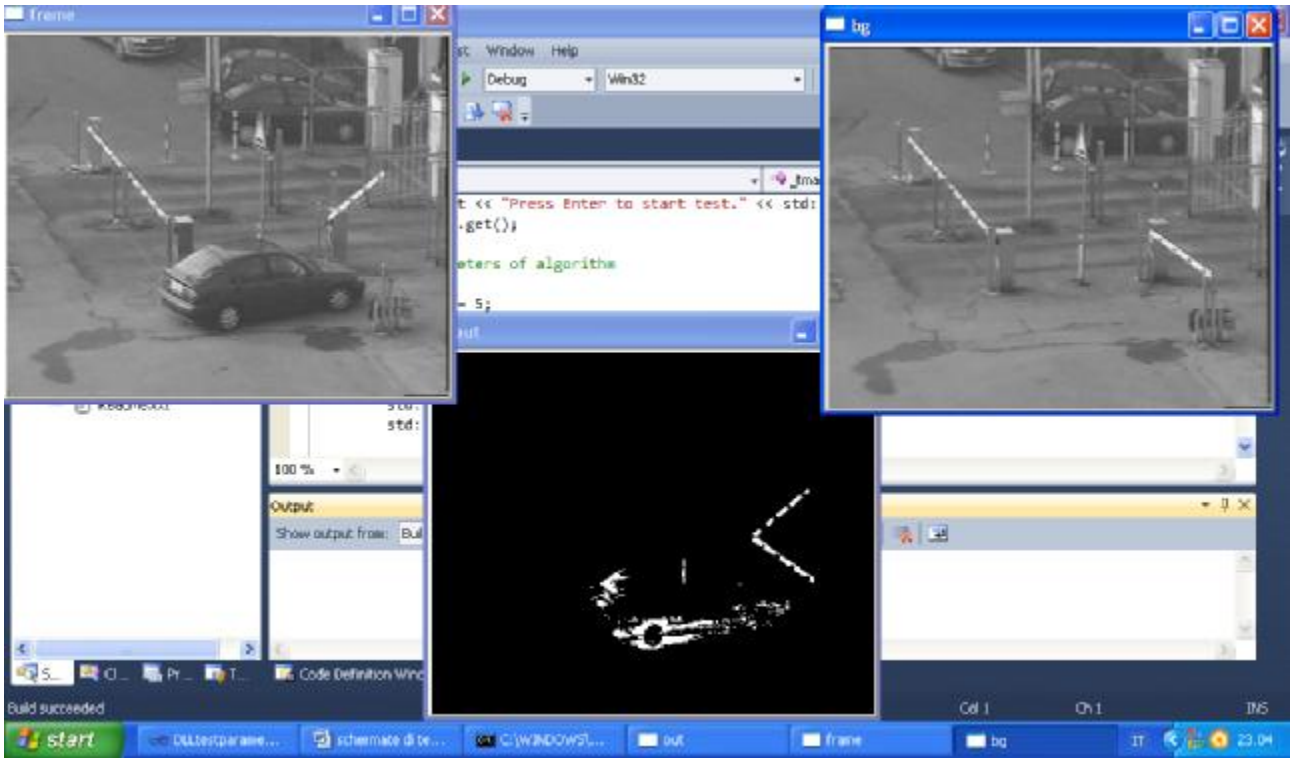




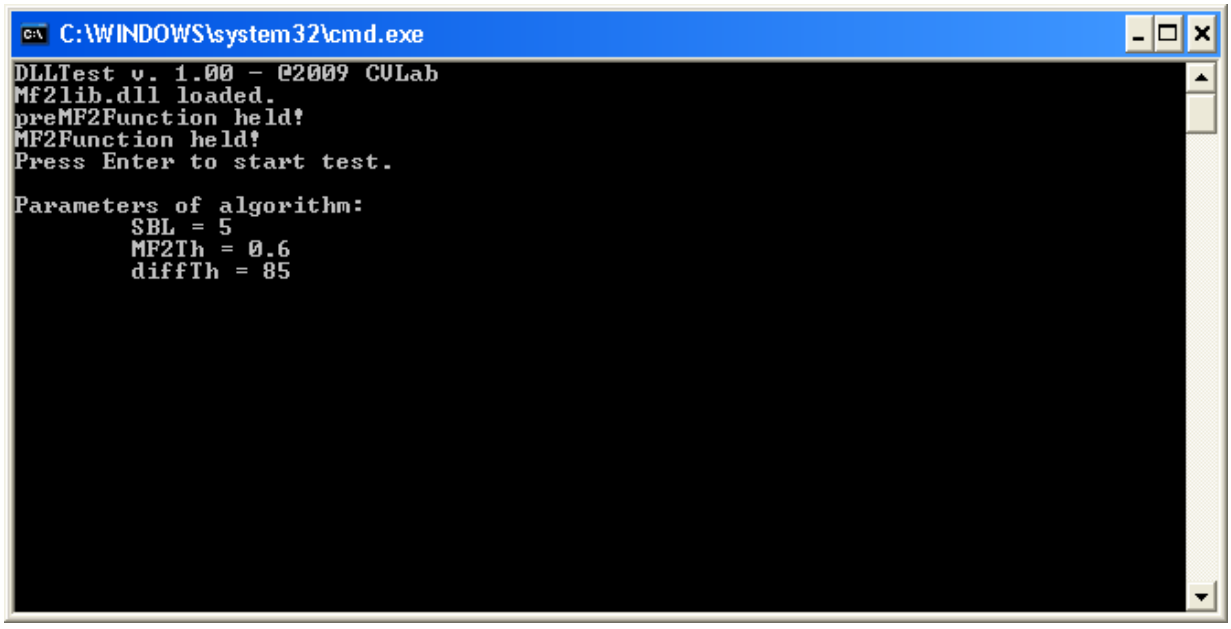
Come si può notare, un valore basso della soglia comporta una precisione piuttosto alta nella segmentazione del foreground, e questo si evidenzia osservando che gli oggetti in movimento appaiono di un bianco abbastanza uniforme nella maschera di modifica finale, e ci sono pochi falsi negativi (ovvero poche zone nere in corrispondenza degli oggetti in movimento) .

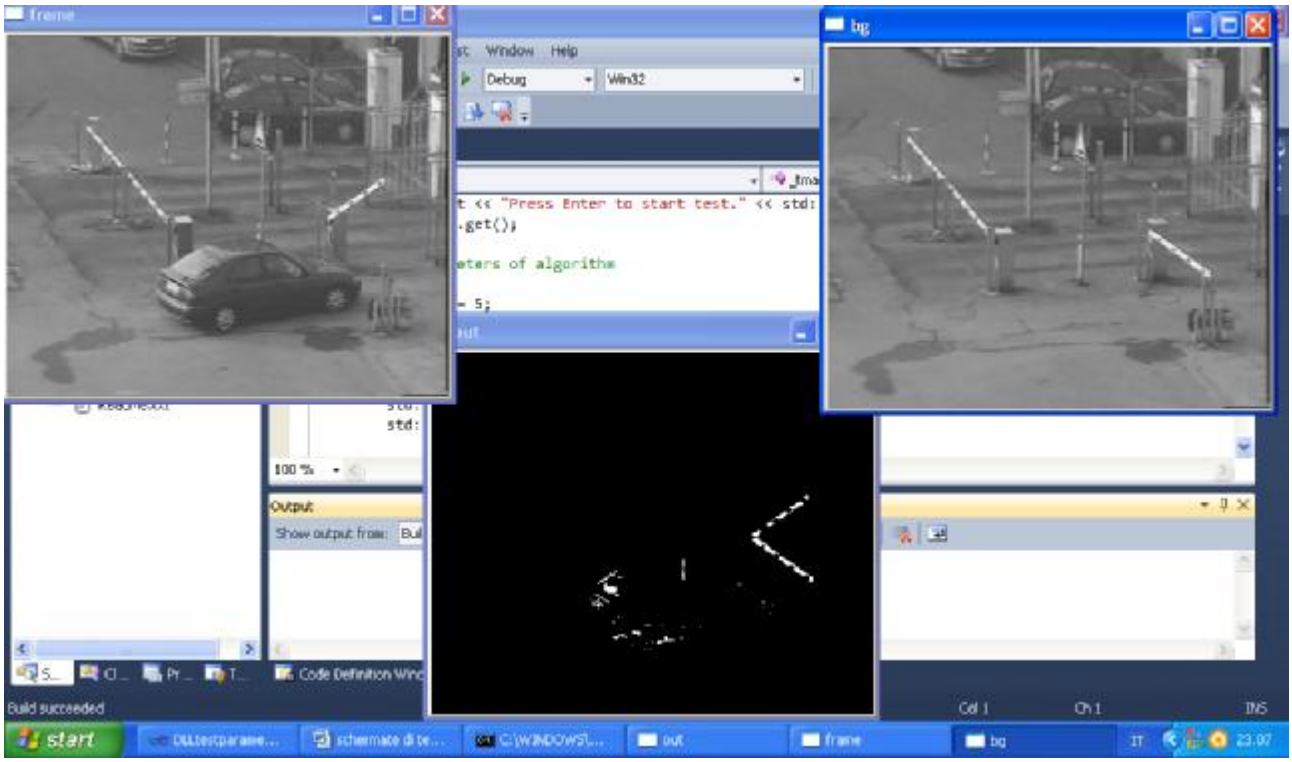
2) Valore soglia pari a 75





3) Valore soglia pari a 85

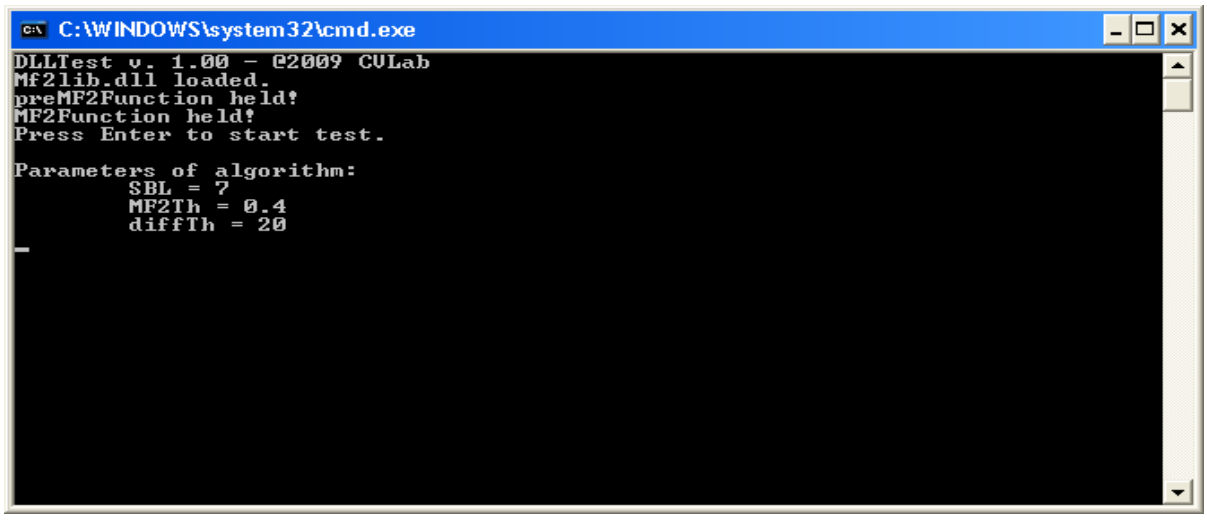


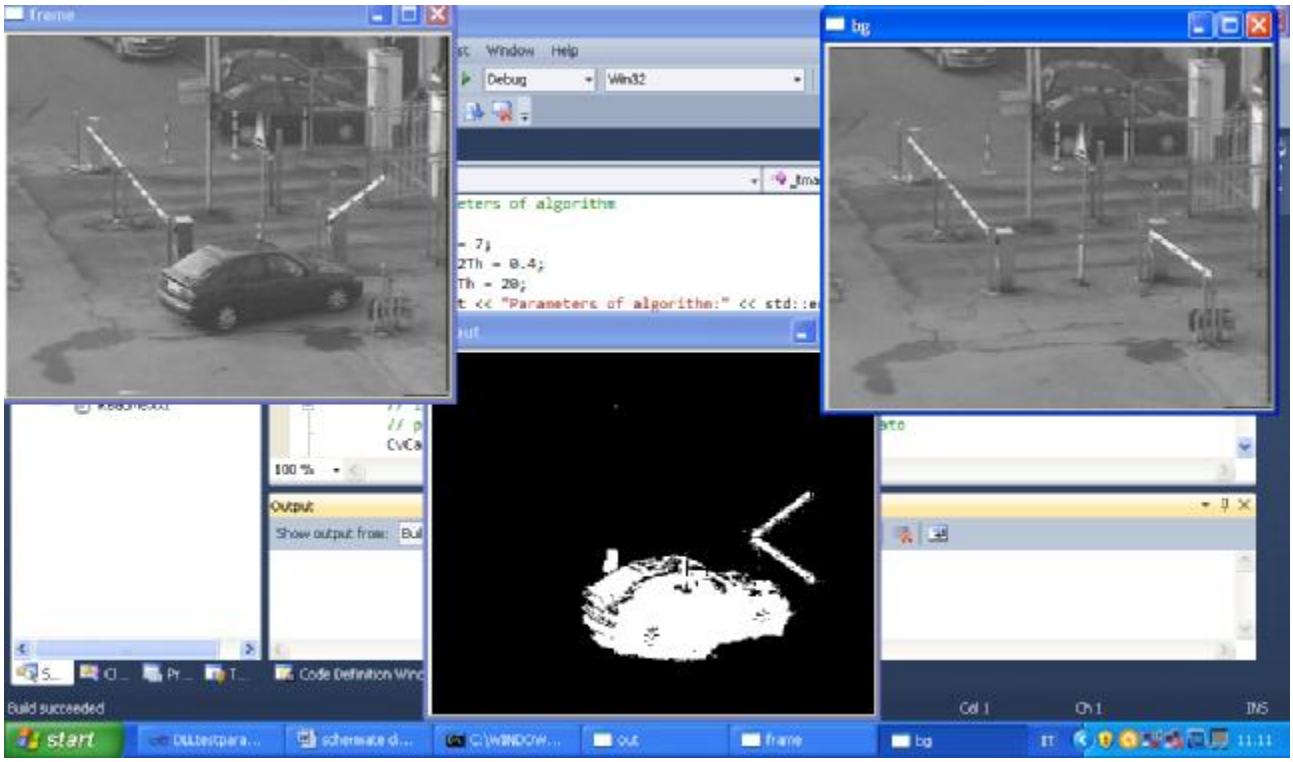


Come si può notare, al crescere della soglia la precisione dell'algoritmo scende.
Se la soglia supera il 90%, la precisione dell'algoritmo scende al di sotto del 50% e questo rende il suo uso assai poco conveniente. Guardando la terza schermata, ove il valore della soglia è pari all'85%, vediamo infatti una change mask dove l'oggetto in movimento (in bianco) appare estremamente sbiadito, è questo significa che ci sono pochi veri positivi (in bianco) ma anche molti falsi negativi (le parti dell'auto in foreground che nelle precedenti schermate erano bianche ed ora invece appaiono nere).

Nei test che seguono, osserviamo invece cosa accade lasciando invariato il valore della soglia finale diffTh e modificando i valori degli altri 2 parametri.

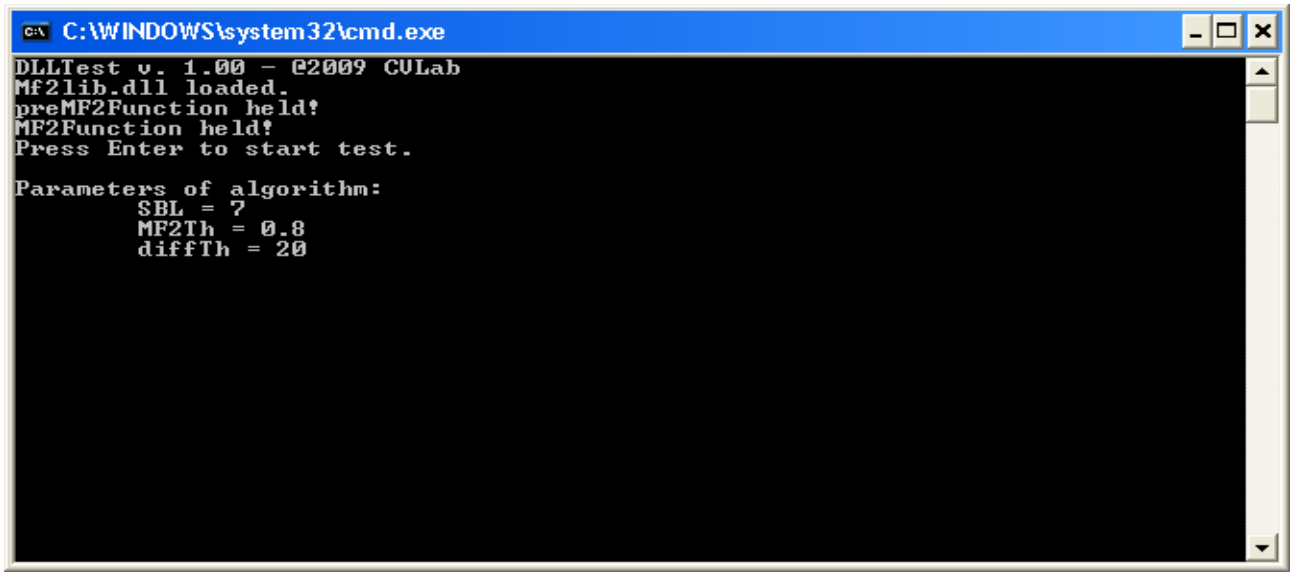
1) aumentiamo la dimensione della finestra in cui effettuare il matching e diminuiamo la soglia intermedia usata per stabilire quali punti vanno inseriti nello sfondo corrente Fb. La soglia finale viene invece lasciata bassa per dare maggior precisione.

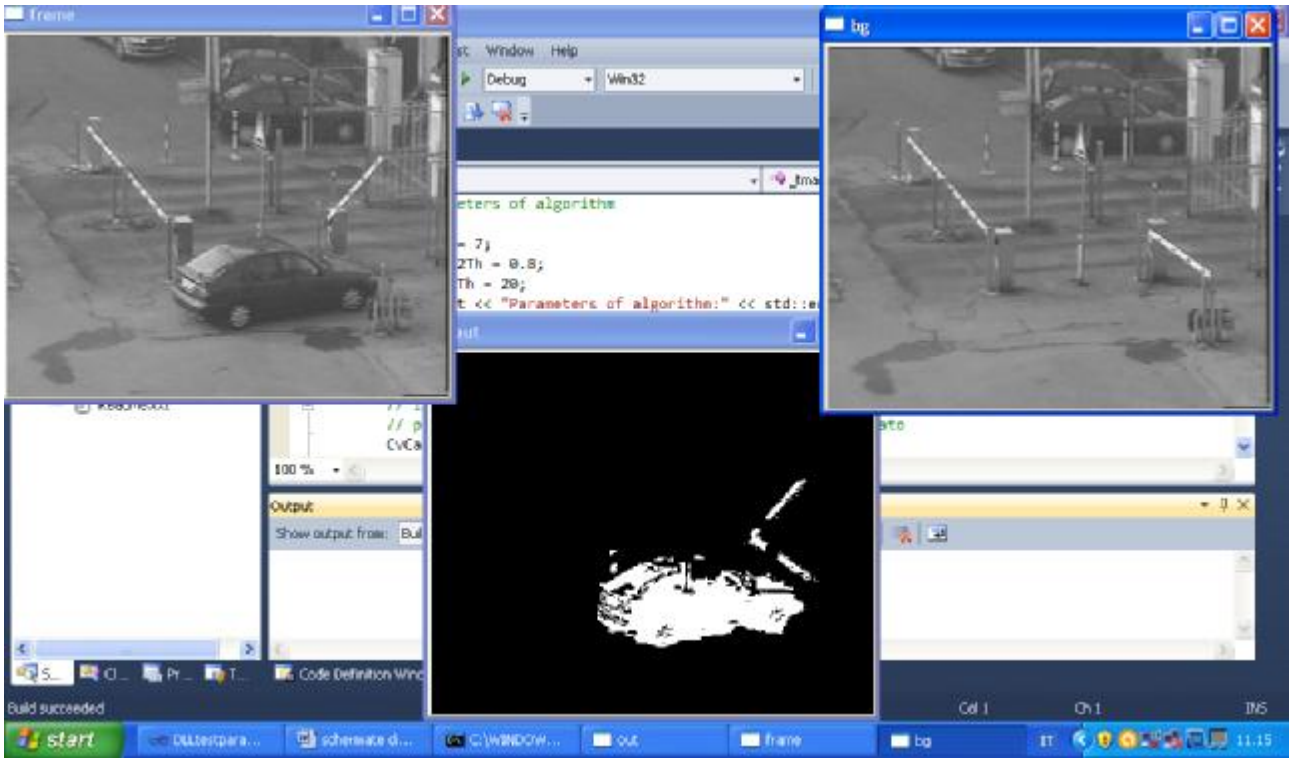




Diminuendo la soglia intermedia diminuiscono anche i falsi negativi e quindi la segmentazione del foreground risulta essere più precisa.

2) aumentiamo la dimensione della finestra in cui effettuare il matching ed aumentiamo la soglia intermedia usata per stabilire quali punti vanno inseriti nello sfondo corrente Fb. La soglia finale viene invece lasciata bassa per dare maggior precisione.





Aumentando la soglia intermedia aumentano anche i falsi negativi e quindi la segmentazione del foreground risulta essere meno precisa

Riferimenti Bibliografici

1 - Sito web del software analizzato nella relazione:
http://vision.deis.unibo.it/software/94-mf-cd

2 - Articolo di riferimento:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.78.894&rep=rep1&type=pdf

3 - Approfondimento sul concetto di change detection:
Alessandro Lanza, Luigi Di Stefano
“ Seminario di Change Detection”
http://unina.stidue.net/Elaborazione%20di%20Segnali%20Multimediali/Materiale/Luigi%20Di%20Stefano/Seminario%20-%20Change%20Detection.pdf

4 - Approfondimento su OpenCV 1.0
Federico Tombari
Introduzione a Opencv
http://www.google.it/url?sa=t&rct=j&q=&esrc=s&source=web&cd=3&ved=0CD4QFjAC&url=http%3A%2F%2Fdidattica.arces.unibo.it%2Ffile.php%2F59%2FElaborazione_dellImmagine_L-S%2FEsercitazioni%2Fopencv.pdf&ei=wU2GU6asDq2b0AWb_oGQDA&usg=AFQjCNEOEy7pXmI8bRj1R7jLlTAmSBdfXQ&bvm=bv.67720277,d.d2k&cad=rja